УДК 004.056

Н. В. Юрина,
магистрант
О. Г. Блинков,
д-р техн. наук
Уральский федеральный университет,
Екатеринбург

РЕАЛИЗАЦИЯ СЕРВЕРА АВТОРИЗАЦИИ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ С ОТКРЫТЫМ ИСХОДНЫМ КОДОМ

Предоставление доступа к защищенным ресурсам при необходимости передавать третьей стороне персональные данные пользователя является проблемой на многих машиностроительных предприятиях. Использование сторонних серверов не всегда надежно и зачастую обходится довольно дорого. Создание собственного сервера авторизации позволит избежать подобных проблем. Разработан сервер авторизации, который позволяет получать необходимую информацию, используя только токен доступа. Он имеет минимальную функциональность, которая в дальнейшем может быть расширена или настроена под нужды конкретного предприятия.

Ключевые слова: C#, ASP.NET Core, OAuth, авторизация, токен доступа.

AN AUTHORIZATION SERVER DEVELOPMENT BASED ON AN OPEN SOURCE LIBRARY

Providing access to protected resources with necessary transfer personal data to a third-party is a problem in many engineering companies. Using off-brand services is not always reliable and often quite expensive. Creating your own authorization server will help to avoid such problems. An authorization server was developed that allows to obtain necessary information, using only an access token. It has minimal functionality that can be expanded or adapted to the needs of a particular company.

Keywords: C#, ASP.NET Core, OAuth, authorization, access token.

Данные и информация — одни из наиболее ценных и важных ресурсов для любой современной компании. Все более часто появляются новости о несанкционированном доступе к тем или иным сайтам и базам данных. Пользователи таких ресурсов несут большие потери в виде своих персональных данных, а плата компании за необеспеченную безопасность ресурсов может оказаться крайне высокой. Именно поэтому нужно обеспечить наиболее защищенный способ передачи персональных данных сторонним приложениям. Один из способов реализации защиты — это создание собственного сервера авторизации, который позволяет предусмотреть необходимые меры безопасности при работе с информацией [1].

Для реализации сервера использован протокол OAuth 2.0, который является стандартным протоколом авторизации, позволяющим выдать одному приложению права на доступ к ресурсам пользователя на другом сервисе [2; 3]. А также библиотека с открытым исходным кодом OpenIddict [4]. OpenIddict — это расширение для платформы ASP.NET Core, которое предназначено для управления потоком запросов аутентификации. Оно хорошо зарекомендовало себя во множестве интернет-проектов.

Общий принцип работы с сервером авторизации:

1. Обращение к серверу за токеном доступа.
2. Поучение защищенных ресурсов, используя полученный ключ.

Результатом авторизации приложения является токен доступа — ключ, предъявление которого является пропуском к защищенным ресурсам. Обращение к ним происходит через HTTP-запрос с указанием в заголовке или в качестве одного из параметров полученного токена доступа.

Создаем новое веб-приложение ASP.NET Core с пустым шаблоном в VisualStudio 2019 [5].

После того как приложение создано, необходимо в консоли диспетчера пакетов набрать следующие команды, чтобы добавить пакеты необходимые для использования библиотеки OpenIddict:

```
>dotnet add package OpenIddict --version 2.0.0-rc3-final [6];
>dotnet add package OpenIddict.EntityFrameworkCore --version 2.0.0-rc2-final [7].
```

В данном проекте используется свободная реляционная система управления базами данных MySQL [8] и объектно-ориентированная технология доступа к данным EntityFrameworkCore [9]. Чтобы взаимодействовать с базой данных через EntityFramework, нужен контекст данных — класс, унаследованный от класса Microsoft.EntityFrameworkCore.DbContext. Поэтому в папку Models добавляется новый класс, который назовем AuthenticationServerContext:

```
1.  public class AuthenticationServerContext:
IdentityDbContext<ApplicationUser>
2.  {
3.  public AuthenticationServerContext
(DbContextOptions options): base(options) {}
4.  }
```

Чтобы подключаться к базе данных, необходимо задать параметры подключения. Для этого изменяется файл appsettings.json с добавлением в него определение строки подключения:

```
5.  {«ConnectionStrings»:{
6.  "AuthenticationServerContext":
"server=127.0.0.1; uid=root; password=1;
database=oauth;"},
7.  }
```

И последним шагом в настройке проекта является изменение файла Startup.cs. В нем надо добавить в метод ConfigureServices() следующие строки:

```
8.  services.AddDbContext
<AuthenticationServerContext>(options =>
9.  {
10.  options.UseMySql
(configuration.GetConnectionString
("AuthenticationServerContext"));
11.  options.UseOpenIddict();
12.  });
```

На следующем шаге необходимо создать фактическую базу данных для контекста базы данных. Для этого сначала добавляем миграцию базы данных:

```
>dotnet ef migrations add AuthenticationServerCon-
text
```

А затем на основе этой миграции выполняем обновление самой базы данных:

```
>dotnet ef database update
```

Как только команда будет выполнена, в базу данных добавится необходимый для работы набор таблиц.

Следующим шагом осуществляем регистрацию промежуточного ПО. Добавим регистрацию сервисов, требуемых OpenIddict. Для этого обновим метод ConfigureServices, чтобы зарегистрировать службы аутентификации.

```
1.  services.AddOpenIddict()
2.  .AddCore(options =>
3.  {
4.  options.UseEntityFrameworkCore().
UseDbContext<AuthenticationServerContext>();
5.  })
6.  .AddServer(options =>
7.  {
8.  options.EnableAuthorizationEndpoint
(«/connect/authorize»)
9.  .EnableTokenEndpoint(«/connect/token»);
10.  options.RegisterScopes
(OpenIdConnectConstants.Scopes.Profile);
11.  options.AllowAuthorizationCodeFlow()
12.  options.SetAccessTokenLifetime(TimeSpan.
FromMinutes(10));
13.  options.UseMvc();
14.  })
15.  .AddValidation();
```

Секция.AddCore позволяет указать параметры подключения базы данных, переопределить стандартные сущности OpenIddict. В секции.AddServer определяют конечные точки приложения, методы получения токена, срок жизни каждого из токенов и данные пользователя, доступные для приложения. В последней секции.AddValidation добавляет стандартную валидацию, которую при необходимости можно сменить на пользовательскую.

Сервер авторизации не будет работать на этом этапе, так как не реализована фактическая конечная точка авторизации. В папку Controllers и добавим класс AuthorizationController и метод Authorize, который будет обрабатывать запрос с маршрутом /connect/authorize, определенный для конечной точки авторизации OpenIddict:

```
1.  [Authorize, HttpGet(«~/connect/authorize»)]
2.  public async Task<IActionResult> Authorize([-
ModelBinder(typeof(OpenIddictMvcBinder))] OpenId-
ConnectRequest request)
3.  {
```

```
4.  var application = await applicationManager.
FindByClientIdAsync(request.ClientId, HttpContext.
RequestAborted);
5.  if (application == null) return error;
6.
7.  return View(new AuthorizeViewModel
8.  {
9.  ApplicationName = application.DisplayName,
10. RequestId = request.RequestId
11. });
12. }
```

Так же реализуем метод с именем Exchange, доступный по маршруту /connect/token для обработки запросов конечной точки токена OpenIddict:

```
1.  [HttpPost("~/connect/token"), Produces
("application/json")]
2.  public async Task<IActionResult> Exchange([-
ModelBinder(typeof(OpenIddictMvcBinder))]
OpenIdConnectRequest request)
3.  {
4.  if (request.IsAuthorizationCodeGrantType())
5.  {
6.  var info = await HttpContext.Authenticate
Async(OpenIddictServerDefaults.
AuthenticationScheme);
7.  var user = await userManager.GetUserAsync
(info.Principal);
8.  if (user == null ||!await signInManager.
CanSignInAsync(user))
9.  {
10. throw new Exception();
11. }
12. var ticket = await CreateTicketAsync(request,
user, info.Properties);
13. return SignIn(ticket.Principal, ticket.Properties,
ticket.AuthenticationScheme);
14. }
15.
16. return BadRequest();
17. }
```

Таким образом, выполнена поставленная задача создания сервера авторизации на основе библиотеки с открытым исходным кодом OpenIddict. Разработана пошаговая инструкция для реализации собственного приложения на платформе ASP.NET Core, приведены необходимые расширения и дополнения. Следует отметить, что в дальнейшем данный сервер может применяться на сайтах для авторизации приложений посредством протокола OAuth. OAuth — простой стандарт авторизации, основанный на базовых принципах Интернета, что делает возможным применение авторизации практически на любой платформе. Сервер реализует базовые методы, необходимые для минимальной работы, которые могут быть расширены или настроены в соответствии с потребностями конкретного предприятия. Стандарт имеет поддержку крупнейших площадок и очевидно, что его популярность будет только расти.

Список литературы

1. The OAuth 2.0 Authorization Framework / ed. D. Hardt. Microsoft, 2012.
2. Microsoft.NET Core 2.0 : [интернет-источник]. — URL: https://dotnet.microsoft.com/download/dotnet-core/2.0 (дата обращения: 10.11.2019).
3. OAuth 2.0 : [интернет-источник]. — URL: https://oauth.net/2/ (дата обращения: 05.11.2019).
4. Openiddict/openiddict-documentation : [интернет-источник]. — URL: https://github.com/openiddict/openiddict-documentation (дата обращения: 10.11.2019).
5. Microsoft | Документация по Visual Studio : [интернет-источник]. — URL: https://docs.microsoft.com/ru-ru/visualstudio/ide/quickstart-aspnet-core?view=vs-2019 (дата обращения: 06.11.2019).
6. OpenIddict.Core : [интернет-источник]. — URL: https://www.nuget.org/packages/OpenIddict.Core/2.0.0-rc3-final (дата обращения: 09.11.2019).
7. OpenIddict.EntityFrameworkCore : [интернет-источник]. — URL: https://www.nuget.org/packages/OpenIddict.EntityFrameworkCore/2.0.0-rc2-final (дата обращения: 01.11.2019).
8. MySQL : [интернет-источник]. — URL: https://www.mysql.com/ (дата обращения: 03.11.2019).
9. Aspnet/EntityFrameworkCore : [интернет-источник]. — URL: https://github.com/aspnet/EntityFrameworkCore (дата обращения: 06.11.2019).